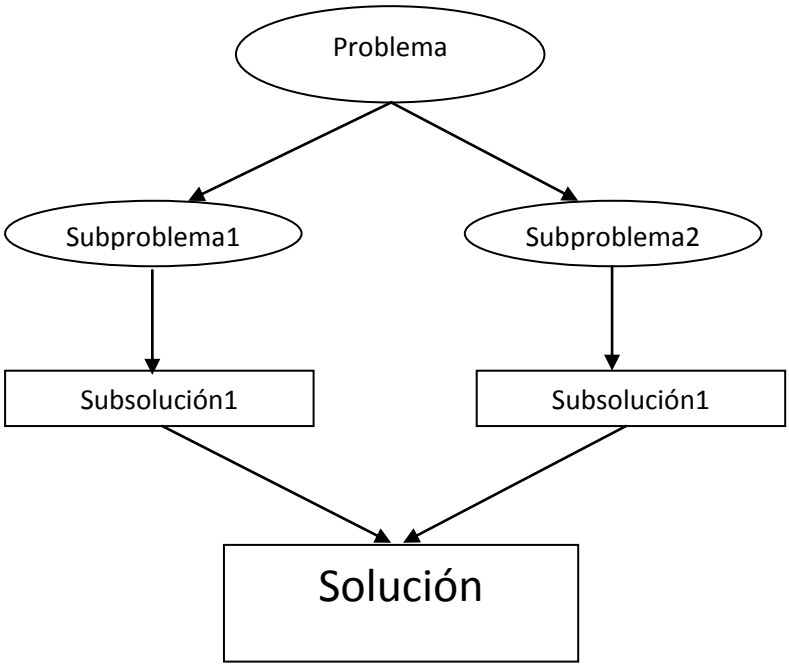
Técnicas de Diseño de Algoritmos

Divide y vencerás



Paradigma de diseño de algoritmos. Si tenemos un problema a resolver (ordenar una lista por ejemplo), dividiremos el problema en dos subproblemas del mismo tipo, que resolveremos obteniendo las subsoluciones a los problemas y luego combinaremos para construir la solución al problema original.

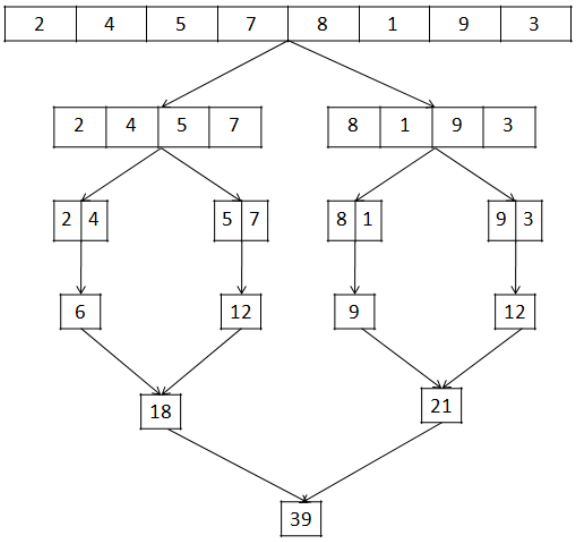
A la hora de resolver cada subproblema nos plantearemos lo mismo, si lo podemos dividir, lo haremos en otros dos problemas más sencillos. En los subsubproblemas lo haremos también y así recursivamente.

El mecanismo es:

- Si el problema no es básico lo dividimos.
- Si es básico:
    - Resolvemos.
    - Combinamos soluciones.

El caso base es aquel que no se puede dividir. Puede haber más de un caso base.

Ejemplo:



Ordenación por mezcla (Merge-Sort)

Material

Transparencias: Sorting Algorithms

ERRORES: Página 13: "If exists c>0 such that **f(n)=O(n)**" cuando debería ser: "**$O(n^c)$**" y, más abajo: "if $k>b^c$ then **$T(n)=O(n^{\log_b n})$**" cuando debería ser: "**$O(n^{\log_b k})$**"

Es una secuencia de una entrada con n elementos que se compone de tres pasos:

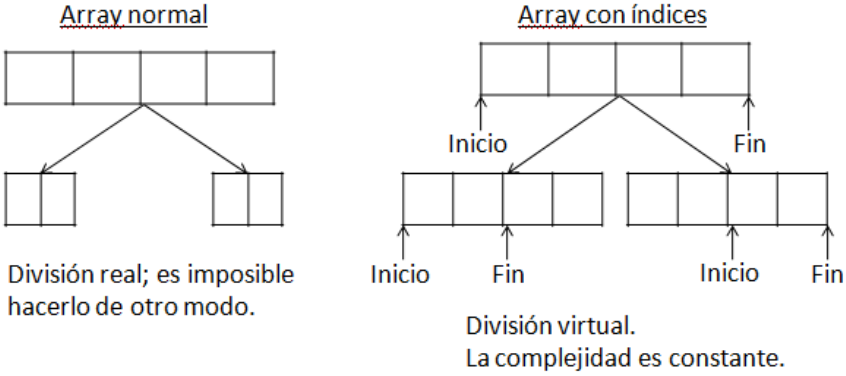
- Dividir: se divide en dos secuencias de más o menos $\frac{n}{2}$ elementos.
- Recursivo: recursivamente ordenar las dos secuencias.
- Vencer: unir s1 y s2 en una única secuencia ordenada. Tiene complejidad O(n)

Todo el método tiene complejidad O(n·Log(n)) que reside casi por completo en la parte de unión de las subsoluciones.

Quick-sort

Es similar a merge-sort. Antes se hacía el trabajo en la mezcla, pero ahora se hará a la hora de dividir. Es decir, la complejidad reside en la división del problema en subproblemas.

La parte mala de este algoritmo es que su caso peor es $O(n^2)$ y no es de tipo divide y vencerás, pero este caso es estadísticamente muy improbable. La complejidad del caso medio es O(n log n). Para conseguir que la división en sí (el corte del problema) tenga complejidad constante



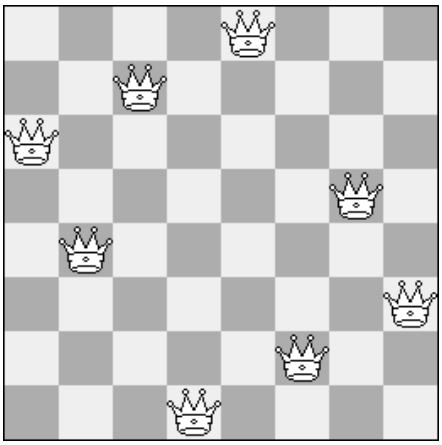
usaremos un array con índices, para practicar un corte virtual.

Se elige un elemento pivote, uno cualquiera en principio. Se dividirá la entrada en elementos menores que el pivote, iguales que el pivote (si los hubiera), y mayores que el pivote. Se resuelve cada uno de los subproblemas, obtenemos las soluciones y concatenamos las tres partes.

Backtracking (Búsqueda por retroceso).

Es un tipo de problemas en los que en general hay una búsqueda en un espacio de posibilidades, se suelen llamar espacio de estados, donde la solución se construye de forma incremental. Debemos aplicar restricciones, **cuanto antes**, para "podar" la búsqueda.

Ejemplo: En un tablero de ajedrez nxn queremos colocar una reina. La primera a colocar tenemos n^2 posibilidades, pero para la siguiente (si no podemos colocar en el mismo) hay n^2-1 posibilidades. Hay una forma para colocar por fuerza bruta que colocamos al azar y después comprobamos si es válida (no hay reinas que se amenacen entre sí). Esto es muy ineficiente, la complejidad para esto es desorbitada.



Para evitar problemas se crean restricciones:

- En cada fila solo se puede poner una reina.
- En cada columna solo se puede poner una reina.
- En cada diagonal solo puede haber una reina.

Así cada vez que pongamos una reina bloquearemos las casillas que no podemos colocar nada más y la complejidad de la búsqueda disminuirá.

Este proceso busca la solución más rápida creando restricciones.

Voraz

Este algoritmo no se verá en profundidad, ni siquiera aparece en las transparencias. Pero solo un pequeño apunte de lo que es un algoritmo de tipo voraz:

Un algoritmo voraz es aquel que, para resolver un determinado problema elige la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento.